

A Constructive Proof On the Compositionality of Linearizability

Haoxiang Lin

Abstract

Linearizability is the strongest correctness property for both shared memory and message passing concurrent systems. One promising nature of linearizability is the compositionality: a history(execution) is linearizable if and only if each object subhistory is linearizable, which is instructive in that we are able to design, implement and test a whole system from the bottom up. In this paper, we propose a new methodology for system model that histories are defined to be special well-ordered structures. The new methodology covers not only finite executions as previous work does, but also infinite ones in reactive systems that never stop. Then, we present a new constructive proof on the compositionality of linearizability inspired by merge sort algorithm.

1 INTRODUCTION

Linearizability [3] is the strongest correctness property for both shared memory and message passing concurrent systems. Informally, a piece of execution(e.g. method call) of one process is linearizable if it appears to take effect *instantaneously* at a moment during its lifetime. This implies that the final effect of concurrent executions of multiple processes is actually *equivalent* to the *sequential* one by some single process. In this sense, linearizability greatly reduces the difficulty to reason about concurrent systems to that on traditional sequential ones.

Additionally, linearizability is *compositional*, which means a concurrent execution is linearizable if and only if each object(component) sub-execution is linearizable. Such a promising nature is instructive in that we are able to design, implement and test a huge system from the bottom up. On the contrary, if compositionality does not hold, *the correctness of a concurrent system has to rely on a centralized scheduler or additional constraints on objects* [3, 4] which makes the system construction much more complicated.

Original definition on linearization [1, 2, 3, 4] models a system execution by a history which is a finite sequence of operation events. This does not capture the inherent ordering between those events, and rules out infinite executions in reactive systems that never stop. Besides, the proof on compositionality in [3]¹ is an existence proof that demonstrates a global partial ordering among individual object subhistories. Although such a partial ordering could be extended to the final linearization(s) by Order-Extension Principle [5], it remains unknown how to *construct* a real linearization fast in practice.

In this paper, we first propose a new methodology for system model that histories are defined to be well-ordered structures which are finitely-partitionable and well-formed in addition. The new methodology covers not only finite executions as previous work does, but also infinite ones in reactive systems as well. Then we strengthen the original definition of linearizability to exclude an intricate case where a pair of method calls are comparable in the complete extension history while one of them is pending in the original history. Finally, inspired by merge sort algorithm, we present a new constructive proof on the compositionality of linearizability.

2 SYSTEM MODEL

2.1 Overview

We adopt basic notations such as *process*, *object* and *event* from [1, 2, 3, 4]. Processes are single-threaded that execute in parallel, and exchange information with each other through shared objects. Every object has some non-overlapped and persistent *memory* for keeping values, an associated *type* defining the value domain, and some primitive *operations* as the only interfaces for object creation and manipulation.

¹Proofs in [1, 2, 4] are not sound. In [1, 2], authors do not consider the *inter-object* partial orderings. In [4], the inductive proof misses a situation that there is no maximal method call among the last ones from each object history.

Definition 2.1. A concurrent system is an ordered pair $\langle P, O \rangle$ comprising a finite set P of $m \in \mathbb{N}$ processes $\{p_1, p_2, \dots, p_m\}$ and a finite set O of $n \in \mathbb{N}$ objects $\{o_1, o_2, \dots, o_n\}$.

Definition 2.2. An object o is an ordered pair $\langle MEM, OP \rangle$ where MEM is the memory and OP is a finite set of $k \in \mathbb{N}$ operations $\{o.op_1, o.op_2, \dots, o.op_k\}$.

An executing instance of object operations is named *method call*. During its lifetime, we regard that unique *events* are generated. E.g. there is an *invocation* event at the very beginning when a method call starts, and a *response* event right at the moment it finishes; while in the middle when logs are outputted to the screen, a *print* event is emitted. Here we only consider invocation and response events because they completely depict method call behavior. Invocation and response events are identified by five key factors: in which process it is generated, which method call it belongs to, on which object the method call manipulates, operation type, and the associated payloads. We assume that the set of method calls that a process executes is **countable**, so that we are able to use natural numbers to index the first four factors. An invocation event is then written as $inv\langle i, j, x, y, args^* \rangle$, and an response event as $resp\langle i, j, x, y, term(res^*) \rangle$. Such notation expresses that the event belongs to the j -th method call made by process p_i on the operation op_y of object o_x . $args^*$ and res^* stand for the arguments and results respectively, while $term$ is the termination condition. An invocation event $inv\langle i, j, x, y, args^* \rangle$ and response event $resp\langle r, s, u, v, term(res^*) \rangle$ are *matching* if $(i = r) \wedge (j = s) \wedge (x = u) \wedge (y = v)$. A method call is thus the ordered pair of its invocation and response events.

In the above we present informal definitions on process, object, operation, event and method call. From pedantic viewpoint, processes are actually instances of program graphs [7]. Objects can be defined inductively from the most basic ingredient of safe, regular or atomic *registers* [9, 10], with the only READ/WRITE/CAS instructions. Operations are mapped to *actions* [11], while invocation/response events are tuples of program location, action as well as the evaluation of the belonging object. Formal definitions assist us to reason more precisely on various correctness and progress properties in a hierarchical way from the essential register executions, however we skip them for being beyond the scope of this paper.

2.2 Histories

In a concurrent system, we consider that processes carry out one method call by another until itself or the system halts. This means a process could not issue another method call if the previous one has not got its response event yet. Executions with such assumption are *well formed*. Precise definition will be presented later, and we exclude non-well formed executions in the sequel. Events in an execution reveal an inherent causal relationship between each other. E.g. An invocation event and its matching response event are reciprocal causation; the same is true between a response event and any later events generated by one process. In fact, this causality constitutes a natural partial ordering:

- (a) (Intra-Process I) An invocation event is the cause of its matching response event.
- (b) (Intra-Process II) Each response event is the cause of its immediate successor invocation event within the same process.
- (c) (Inter-Process) if process p_i sends a message to p_j , then the invocation event of *Send* method call is the cause of the response event of *Receive* method call.
- (d) (Transitivity) if event e_1 is the cause of e_2 and e_2 is the cause of e_3 , then e_1 is the cause of e_3 .

We assume that the events and the partial ordering(causality) have the following characteristics. First, each event is generated by the only process. Second, from process viewpoint, events generated by one process are countable and well ordered isomorphic to a set of natural numbers. Finally, due to the finiteness of processes, such partial ordering is actually a partial well ordering since it is also well-founded [6]. It is beneficial to extend this causality order to a well ordering because we are able to *compare* every two events and find a *least* element from any non-empty event subset. Theorem 2.3 will give a simplified proof on the possibility of such extension. For general cases, please refer to [12]. In practice, if the system has a global high-precision clock, the event generation time could be leveraged to establish the well ordering.

Theorem 2.3. Assume $\langle E, \prec \rangle$ is a partially ordered countable structure that:

- (a) There exists a finite partition $\Pi = \{E_1, E_2, \dots, E_m\}$ of E .

- (b) For each $E_i \in \Pi$, $\prec \cap (E_i \times E_i)$ is a well ordering.

Then \prec can be extended to a well ordering.

Proof. By Order-Extension Principle [5], \prec can be extended to a linear ordering \prec' on E . We claim that \prec' itself is the desired well ordering. For each $E_i \in \Pi$, $\prec' \cap (E_i \times E_i) = \prec \cap (E_i \times E_i)$ since the latter is already a well ordering. Let A be a non-empty subset of E . $\Pi' = \{A_i \mid (A_i = A \cap E_i) \wedge (A_i \text{ is not empty})\}$ is a partition of A . Assume $|\Pi'| = k \in [1, m]$. In each part of A , there exists a least element $e_{j \in [1, k]}$ under \prec' . The set $\{e_1, e_2, \dots, e_k\}$ is a non-empty finite set, so it has a least element e^* which is apparently the least of A . \square

Now we model an execution of a concurrent system by a *history*:

Definition 2.4. A history \mathbf{H} is a well-ordered countable structure $\langle E, \prec \rangle$ of invocation and response events such that:

- (a) (Finitely-Partitionable) There exists a finite partition $\Pi = \{E_1, E_2, \dots, E_m\}$ of E where events in each E_i share the same process id (generated by one process).
- (b) (Well-Formed) For each $E_i \in \Pi$:
 - i The least element of E_i is an invocation event.
 - ii If $e \prec e'$ and e is an invocation event, then e' is the matching response event.
 - iii If $e \prec e'$ and e is a response event, then e' is an invocation event.

Countability of the event set E covers both finite and infinite executions. In the rest of this paper, we use simplified notations for brevity. *inv* and *resp* stand for an invocation event and a response events respectively. e or e' is either type of events. If the event payload inessential to the discussion, ' $_$ ' will be used instead. m represents a method call. $inv(m)$ and $resp(m)$ are m 's invocation and response events respectively. $m(e)$ is e 's belonging method call. \mathbf{H}, \mathbf{H}' are histories. $E_{\mathbf{H}}$ is the event set of \mathbf{H} , while $\prec_{\mathbf{H}}$ is the corresponding well ordering on $E_{\mathbf{H}}$. We say $e \in \mathbf{H}$ if $e \in E_{\mathbf{H}}$, and $m \in \mathbf{H}$ if $inv(m) \in E_{\mathbf{H}}$ and $resp(m) \in E_{\mathbf{H}}$. o stands for an object, and p for a process.

Two histories \mathbf{H} and \mathbf{H}' are *equal* if $E_{\mathbf{H}} = E_{\mathbf{H}'}$ and $\prec_{\mathbf{H}} = \prec_{\mathbf{H}'}$. \mathbf{H} is a *subhistory* of history \mathbf{H}' if \mathbf{H} is a history and $E_{\mathbf{H}} \subseteq E_{\mathbf{H}'} \wedge \prec_{\mathbf{H}} \subseteq \prec_{\mathbf{H}'}$. This ensures that their event partitions are also compatible. Conversely \mathbf{H}' is called an *extension* of \mathbf{H} . We denote such subhistory-extension relation as $\mathbf{H} \subseteq \mathbf{H}'$. For a history \mathbf{H}' , if $E \subseteq E_{\mathbf{H}'}$, then $\mathbf{H} = \langle E, \prec_{\mathbf{H}'} \cap (E \times E) \rangle$ is a subhistory of \mathbf{H}' because $\prec_{\mathbf{H}'} \cap (E \times E)$ is a well ordering [6] and \mathbf{H} apparently meets the two requirements of history definition. For two histories $\mathbf{H} \subseteq \mathbf{H}'$, we define their *difference* $\mathbf{H}' - \mathbf{H} = \langle (E_{\mathbf{H}'} \setminus E_{\mathbf{H}}), \prec_{\mathbf{H}'} \cap ((E_{\mathbf{H}'} \setminus E_{\mathbf{H}}) \times (E_{\mathbf{H}'} \setminus E_{\mathbf{H}})) \rangle$. $\mathbf{H}' - \mathbf{H}$ is a history too. An invocation event is *pending* if its matching response event does not exist in the history. A method call is *pending* if its invocation event is pending. A method call is *complete* if both its invocation and response events are in the history. $\text{complete}(\mathbf{H})$ is the maximal subhistory of \mathbf{H} without any pending invocation events. A history \mathbf{H} is *complete* if $\text{complete}(\mathbf{H}) = \mathbf{H}$.

From the well ordering on events, we are able to induce a partial ordering on method calls such that such that $m \prec_{\text{MethodCall}} m'$ if $resp(m) \prec inv(m')$. Here m' may be pending. Two method calls are *concurrent* if neither method call's response event (if there exists) precedes the invocation event of the other. A history \mathbf{H} is *sequential* if:

- (a) The least event of \mathbf{H} is an invocation event.
- (b) If $e \prec e'$ and e is an invocation event, then e' is the matching response event.
- (c) If $e \prec e'$ and e is a response event, then e' is an invocation event.

It is obvious that there will be at most one pending method call in a sequential history, and the partial ordering on method calls is actually a well ordering. A history is *concurrent* if it is not sequential.

A *process subhistory* $\mathbf{H}|p$ is the maximal subhistory of \mathbf{H} in which all events are generated by process p . Then the well-formation requirement of history definition can be restated as: every process subhistory is sequential. An *object subhistory* $\mathbf{H}|o$ is defined similarly for an object o . \mathbf{H} is a *single-object* history if there exists o such that $\mathbf{H}|o = \mathbf{H}$. Such single-object history is written as \mathbf{H}_o . Two histories \mathbf{H} and \mathbf{H}' are *equivalent* if for each process p , $\mathbf{H}|p = \mathbf{H}'|p$.

We adopt the techniques in [3] to define the correctness of histories. \mathbf{H} is a *prefix* of \mathbf{H}' if $\mathbf{H} \subseteq \mathbf{H}'$ and for all events $e \in \mathbf{H}$ and $e' \in \mathbf{H}' - \mathbf{H}$ it holds that $e \prec_{\mathbf{H}'} e'$. A set of histories is *prefix-closed* if whenever \mathbf{H} is in the set, every prefix of \mathbf{H} are also the members. A *sequential specification* for an object is a prefix-closed set of single-object sequential histories for that object. A history \mathbf{H} is *legal* if it is sequential and for each object o , $\mathbf{H}|o$ belongs to o 's sequential specification.

2.3 Properties of History

In this subsection we deduce some properties of history for later use in proving the compositionality of linearizability.

Proposition 2.5. Two histories \mathbf{H} and \mathbf{H}' are equal if

- (a) $e \in \mathbf{H} \Leftrightarrow e \in \mathbf{H}'$
- (b) $e \prec_{\mathbf{H}} e' \Leftrightarrow e \prec_{\mathbf{H}'} e'$

Proof. This is a restatement of history equality. Condition 1 implies $E_{\mathbf{H}} = E_{\mathbf{H}'}$, while condition 2 implies $\prec_{\mathbf{H}} = \prec_{\mathbf{H}'}$. \square

Proposition 2.6. Two sequential and complete histories \mathbf{H} and \mathbf{H}' are equal if

- (a) $m \in \mathbf{H} \Leftrightarrow m \in \mathbf{H}'$
- (b) $m \prec_{\mathbf{H}} m' \Leftrightarrow m \prec_{\mathbf{H}'} m'$

Proof. 1.

$$\begin{aligned} e \in \mathbf{H} &\Leftrightarrow (m(e) \text{ is complete}) \wedge (m(e) \in \mathbf{H}) \\ &\Leftrightarrow m(e) \in \mathbf{H}' \\ &\Leftrightarrow e \in \mathbf{H}' \end{aligned}$$

2. Assume $e \prec_{\mathbf{H}} e'$, then there are two cases:

- i $m(e) = m(e')$. That is, e and e' are matching events. Then $m(e)$ is also in \mathbf{H}' and $e \prec_{\mathbf{H}'} e'$.
- ii $m(e) \neq m(e')$. That is, e and e' belong to two different complete method calls. There are four sub-cases on the types of e and e' . In each sub-case we have $m \prec_{\mathbf{H}} m'$ since a response event is the immediate successor of its matching invocation event in a sequential history. By condition (b), we get $m \prec_{\mathbf{H}'} m'$ and then $e \prec_{\mathbf{H}'} e'$.

In both cases we have $e \prec_{\mathbf{H}} e' \Rightarrow e \prec_{\mathbf{H}'} e'$. Similarly we have $e \prec_{\mathbf{H}'} e' \Rightarrow e \prec_{\mathbf{H}} e'$.

By Proposition 2.5, we conclude that \mathbf{H} and \mathbf{H}' are equal. \square

Proposition 2.7. $\text{complete}(\mathbf{H})$ is complete.

Proof. $\text{complete}(\mathbf{H})$ does not have pending invocation events any more, therefore its maximal subhistory without pending invocation events is itself. \square

Proposition 2.8. If \mathbf{H} is complete, then both $\mathbf{H}|o$ and $\mathbf{H}|p$ are complete.

Proof. $\mathbf{H}|o$ and $\mathbf{H}|p$ do not have any pending invocation events. \square

Proposition 2.9. If \mathbf{H} and \mathbf{H}' are equivalent, then $m \in \mathbf{H} \Leftrightarrow m \in \mathbf{H}'$.

Proof.

$$\begin{aligned} m \in \mathbf{H} &\Leftrightarrow \exists p (m \in \mathbf{H}|p) \\ &\Leftrightarrow \exists p (m \in \mathbf{H}'|p) \\ &\Leftrightarrow m \in \mathbf{H}' \end{aligned}$$

\square

Proposition 2.10. If $\mathbf{H} \subseteq \mathbf{H}'$ and $e, e' \in \mathbf{H}$, then $e \prec_{\mathbf{H}} e' \Leftrightarrow e \prec_{\mathbf{H}'} e'$.

Proof. By definition of subhistory. \square

Proposition 2.11. If $\mathbf{H} \subseteq \mathbf{H}'$ and $m, m' \in \mathbf{H}$, then $m \prec_{\mathbf{H}} m' \Leftrightarrow m \prec_{\mathbf{H}'} m'$.

Proof. By Proposition 2.10. \square

Proposition 2.12. If $\mathbf{H} \subseteq \mathbf{H}'$, then $(\mathbf{H}' - \mathbf{H})|o = \mathbf{H}'|o - \mathbf{H}|o$.

Proof. 1. It is obvious that $\mathbf{H}|o \subseteq \mathbf{H}'|o$.

2.

$$\begin{aligned} e\langle i, j, x, y, _ \rangle \in (\mathbf{H}' - \mathbf{H})|o &\Leftrightarrow (o_x = o) \wedge (e\langle i, j, x, y, _ \rangle \in \mathbf{H}') \wedge (e\langle i, j, x, y, _ \rangle \notin \mathbf{H}) \\ &\Leftrightarrow (e\langle i, j, x, y, _ \rangle \in \mathbf{H}'|o) \wedge (e\langle i, j, x, y, _ \rangle \notin \mathbf{H}|o) \\ &\Leftrightarrow e\langle i, j, x, y, _ \rangle \in \mathbf{H}'|o - \mathbf{H}|o \end{aligned}$$

3.

$$\begin{aligned} e \prec_{(\mathbf{H}' - \mathbf{H})|o} e' &\Leftrightarrow (e, e' \in (\mathbf{H}' - \mathbf{H})|o) \wedge (e \prec_{\mathbf{H}' - \mathbf{H}} e') \\ &\Leftrightarrow (e, e' \in \mathbf{H}'|o - \mathbf{H}|o) \wedge (e \prec_{\mathbf{H}' - \mathbf{H}} e') \\ &\Leftrightarrow (e, e' \in \mathbf{H}'|o) \wedge (e, e' \notin \mathbf{H}|o) \wedge (e \prec_{\mathbf{H}'} e') \\ &\Leftrightarrow (e, e' \in \mathbf{H}'|o) \wedge (e, e' \notin \mathbf{H}|o) \wedge (e \prec_{\mathbf{H}'|o} e') \\ &\Leftrightarrow e \prec_{\mathbf{H}'|o - \mathbf{H}|o} e' \end{aligned}$$

□

Proposition 2.13. Assume $\mathbf{H} \subseteq \mathbf{H}'$, $\mathbf{H}' - \mathbf{H}$ contains only response events and e, e' are invocation events, then:

$$(a) \ e \in \mathbf{H} \Leftrightarrow e \in \text{complete}(\mathbf{H}) \Leftrightarrow e \in \mathbf{H}' \Leftrightarrow e \in \text{complete}(\mathbf{H}')$$

$$(b) \ e \prec_{\mathbf{H}} e' \Leftrightarrow e \prec_{\text{complete}(\mathbf{H})} e' \Leftrightarrow e \prec_{\mathbf{H}'} e' \Leftrightarrow e \prec_{\text{complete}(\mathbf{H}')} e'$$

Proof. $\text{complete}(\mathbf{H})$, \mathbf{H}' and $\text{complete}(\mathbf{H}')$ do not affect the orders between invocation events of the original \mathbf{H} . □

Proposition 2.14. $(\mathbf{H}|o)|p = (\mathbf{H}|p)|o$

Proof. 1.

$$\begin{aligned} e\langle i, j, x, y, _ \rangle \in (\mathbf{H}|o)|p &\Leftrightarrow (p_i = p) \wedge (e\langle i, j, x, y, _ \rangle \in \mathbf{H}|o) \\ &\Leftrightarrow (p_i = p) \wedge (o_x = o) \wedge (e\langle i, j, x, y, _ \rangle \in \mathbf{H}) \\ &\Leftrightarrow (e\langle i, j, x, y, _ \rangle \in \mathbf{H}|p) \wedge (o_x = o) \\ &\Leftrightarrow e\langle i, j, x, y, _ \rangle \in (\mathbf{H}|p)|o \end{aligned}$$

2.

$$\begin{aligned} e \prec_{(\mathbf{H}|o)|p} e' &\Leftrightarrow e \prec_{\mathbf{H}|o} e' \Leftrightarrow e \prec_{\mathbf{H}} e' \\ &\Leftrightarrow e \prec_{\mathbf{H}|p} e' \Leftrightarrow e \prec_{(\mathbf{H}|p)|o} e' \end{aligned}$$

Therefore $(\mathbf{H}|o)|p = (\mathbf{H}|p)|o$. □

Proposition 2.15. $\text{complete}(\mathbf{H}|o) = \text{complete}(\mathbf{H})|o$.

Proof. 1.

$$\begin{aligned} e\langle i, j, x, y, _ \rangle \in \text{complete}(\mathbf{H}|o) &\Leftrightarrow (m(e) \text{ is complete}) \wedge (o_x = o) \\ &\Leftrightarrow (e\langle i, j, x, y, _ \rangle \in \text{complete}(\mathbf{H})) \wedge (o_x = o) \\ &\Leftrightarrow e\langle i, j, x, y, _ \rangle \in \text{complete}(\mathbf{H})|o \end{aligned}$$

2. Because $\text{complete}(\mathbf{H})$ and $\text{complete}(\mathbf{H}|o)$ are also subhistories of \mathbf{H} , we have

$$\begin{aligned} e \prec_{\text{complete}(\mathbf{H}|o)} e' &\Leftrightarrow e \prec_{\mathbf{H}|o} e' \Leftrightarrow e \prec_{\mathbf{H}} e' \\ &\Leftrightarrow e \prec_{\text{complete}(\mathbf{H})} e' \Leftrightarrow e \prec_{\text{complete}(\mathbf{H})|o} e' \end{aligned}$$

Therefore $\text{complete}(\mathbf{H}|o) = \text{complete}(\mathbf{H})|o$. □

3 DEFINITION OF LINEARIZABILITY

Definition 3.1. A history $\mathbf{H} = \langle E, \prec \rangle$ is *linearizable* if:

L1: \mathbf{H} can be extended to a new history $\mathbf{H}' = \langle E', \prec' \rangle$ such that $\mathbf{H}' - \mathbf{H}$ contains only response events.

L2: $\text{complete}(\mathbf{H}')$ is equivalent to some legal sequential history \mathbf{S} .

L3: For two different method calls m and m' , if $m \prec_{\text{complete}(\mathbf{H}')} m'$, then $m \prec_{\mathbf{S}} m'$.

Pending method calls in the original history may or may not have taken effects. Those having taken effects are captured by extending \mathbf{H} with their future matching response events. Later restriction to $\text{complete}(\mathbf{H}')$ eliminates the remaining ones without real impact on the system, and that is why we use $\prec_{\text{complete}(\mathbf{H}')}$ instead of $\prec_{\mathbf{H}}$ [3] in condition **L3**.

The above legal sequential history \mathbf{S} is called a *linearization* of \mathbf{H} . \mathbf{H} may have more than one linearization. E.g. if two concurrent complete method calls m, m' operate on different objects, either one precedes the other could be legal in the final linearization. A *linearizable object* is one whose concurrent histories are all linearizable according to its certain sequential specification.

4 COMPOSITIONALITY OF LINEARIZABILITY

Linearizability is compositional in that the system as a whole is linearizable whenever each individual object is linearizable :

Theorem 4.1. \mathbf{H} is linearizable if and only if, for each object o , $\mathbf{H}|o$ is linearizable.

Theorem 4.1 has two parts:

Lemma 4.2. If \mathbf{H} is linearizable, then for each object o , $\mathbf{H}|o$ is linearizable.

Proof. Suppose \mathbf{S} is one linearization of history \mathbf{H} , and \mathbf{H}' is the corresponding extension. We claim that $\mathbf{S}|o$ is an linearization of $\mathbf{H}|o$ whose extension is just $\mathbf{H}'|o$.

1. $\mathbf{H}'|o$ is a history since \mathbf{H}' is a history, and it is obviously that $\mathbf{H}|o \subseteq \mathbf{H}'|o$. Since $\mathbf{H}'|o - \mathbf{H}|o = (\mathbf{H}' - \mathbf{H})|o$ by Proposition 2.12 and $\mathbf{H}' - \mathbf{H}$ contains only response events, $\mathbf{H}'|o - \mathbf{H}|o$ contains only response events too.
2. \mathbf{S} is a legal, sequential and complete history, therefore $\mathbf{S}|o$ is also a legal, sequential and complete history.
3. Because $\text{complete}(\mathbf{H}')$ is equivalent to \mathbf{S} , $\text{complete}(\mathbf{H}')|p = \mathbf{S}|p$.
- 4.

$$\begin{aligned}
 \text{complete}(\mathbf{H}'|o)|p &= (\text{complete}(\mathbf{H}'))|o|p \quad \text{by Proposition 2.15} \\
 &= (\text{complete}(\mathbf{H}'))|p|o \quad \text{by Proposition 2.14} \\
 &= (\mathbf{S}|p)|o \\
 &= (\mathbf{S}|o)|p
 \end{aligned}$$

Thus $\text{complete}(\mathbf{H}'|o)$ is equivalent to $\mathbf{S}|o$.

5.

$$\begin{aligned}
 m \prec_{\text{complete}(\mathbf{H}'|o)} m' &\Rightarrow m \prec_{\text{complete}(\mathbf{H}')|o} m' \\
 &\Rightarrow m \prec_{\text{complete}(\mathbf{H}')} m' \\
 &\Rightarrow m \prec_{\mathbf{S}} m' \\
 &\Rightarrow m \prec_{\mathbf{S}|o} m' \quad \text{since } m, m' \in \mathbf{S}|o
 \end{aligned}$$

□

Lemma 4.3. If for each object o , $\mathbf{H}|o$ is linearizable, then \mathbf{H} is linearizable.

Proof. Let $(\mathbf{H}|o)'$ be an extension of $\mathbf{H}|o$ such that $(\mathbf{H}|o)' - \mathbf{H}|o$ contains only response events, and \mathbf{S}_o be the corresponding linearization. The informal idea is to construct a sequential history $\mathbf{S} = \langle E_{\mathbf{S}}, \prec_{\mathbf{S}} \rangle$ from each \mathbf{S}_o by the following algorithm similar to merge sort:

- a. At the beginning, both $E_{\mathbf{S}}$ and $\prec_{\mathbf{S}}$ are \emptyset .
- b. From the finite set $\{m \mid m \text{ is the least method call of some } \mathbf{S}_o\}$, select a method call $m \in \mathbf{S}_{o_i}$ whose invocation event is minimal in the original history \mathbf{H} . This is feasible because invocation events in each \mathbf{S}_o are also in \mathbf{H} by Proposition 2.13 and events in \mathbf{H} are already well-ordered.
- c. Remove m from \mathbf{S}_{o_i} and append it to \mathbf{S} , that is:

$$\begin{aligned} \prec_{\mathbf{S}} &:= \prec_{\mathbf{S}} \cup \{\langle e, \text{inv}(m) \rangle \mid e \in E_{\mathbf{S}}\} \cup \{\langle e, \text{resp}(m) \rangle \mid e \in E_{\mathbf{S}}\} \\ &\quad \cup \{\langle \text{inv}(m), \text{resp}(m) \rangle\} \\ E_{\mathbf{S}} &:= E_{\mathbf{S}} \cup \{\text{inv}(m), \text{resp}(m)\} \end{aligned}$$

- d. Go back to step b.

Base on the above algorithm, we present the formal construction for both finite and infinite histories:

$$\begin{aligned} E_{\mathbf{S}} &= \bigcup_o E_{\mathbf{S}_o} \\ \prec_{\mathbf{S}} &= \left(\bigcup_o \prec_{\mathbf{S}_o} \right) \cup \{ \langle e, e' \rangle \mid \exists m, m', o_i, o_j \\ &\quad ((e \in m \in \mathbf{S}_{o_i}) \wedge (e' \in m' \in \mathbf{S}_{o_j}) \wedge (i \neq j) \wedge (\text{inv}(m) \prec_{\mathbf{H}} \text{inv}(m'))) \} \end{aligned}$$

Beware the fact that events in $(\mathbf{H}|o_i)' - \mathbf{H}|o_i$ have no causality with those in both $(\mathbf{H}|o_j)' - \mathbf{H}|o_j$ and $\mathbf{H}|o_j$ if $i \neq j$. Therefore we could give them an arbitrary ordering. Now we construct the extension history \mathbf{H}' :

$$\begin{aligned} E_{\mathbf{H}'} &= \bigcup_o E_{(\mathbf{H}|o)'} \\ \prec_{\mathbf{H}'} &= \left(\bigcup_o \prec_{(\mathbf{H}|o)'} \right) \cup \{ \langle e, e' \rangle \mid (e \in \mathbf{H}|o_i) \wedge (e' \in (\mathbf{H}|o_j)' - \mathbf{H}|o_j) \wedge (i \neq j) \} \\ &\quad \cup \{ \langle e, e' \rangle \mid (e \in (\mathbf{H}|o_i)' - \mathbf{H}|o_i) \wedge (e' \in (\mathbf{H}|o_j)' - \mathbf{H}|o_j) \wedge (i < j) \} \end{aligned}$$

We have the following facts on the constructed \mathbf{S} and \mathbf{H}' :

1. \mathbf{S} is a history. It is obvious that $\prec_{\mathbf{S}}$ is a linear ordering. Suppose A is a non-empty subset of $E_{\mathbf{S}}$. Then $\Pi = \{A_i \mid (A_i = A \cap E_{\mathbf{S}_{o_i}}) \wedge (A_i \text{ is not empty})\}$ is a partition of A because all $E_{\mathbf{S}_{o_i}}$ are pair-wisely disjoint. The set $\{e \mid e = \text{inv}(m(e_i)) \text{ where } e_i \text{ is the least of } A_i\}$ does exist since each A_i is well-ordered under $\prec_{\mathbf{S}_{o_i}}$. Let e^* be the least event of such finite set. Then, either e^* or its matching response event is the least element of A depending on whether $e^* \in A$ or not. Thus $\prec_{\mathbf{S}}$ is a well ordering.

Similarly, \mathbf{H}' is a history too.

2. $\mathbf{S}|o = \mathbf{S}_o$ and $\mathbf{H}'|o = (\mathbf{H}|o)'$
3. $m \in \mathbf{S}$ if and only if there exists an object o such that $m \in \mathbf{S}_o$
4. $m \in \text{complete}(\mathbf{H}')$ if and only if $m \in \mathbf{S}$ because:

$$\begin{aligned} m \in \text{complete}(\mathbf{H}') &\Leftrightarrow \exists o (m \in \text{complete}(\mathbf{H}')|o) \\ &\Leftrightarrow \exists o (m \in \text{complete}(\mathbf{H}'|o)) \quad \text{by Proposition 2.15} \\ &\Leftrightarrow \exists o (m \in \text{complete}((\mathbf{H}|o)')) \\ &\Leftrightarrow \exists o (m \in \mathbf{S}_o) \quad \text{since } \text{complete}((\mathbf{H}|o)') \text{ and } \mathbf{S}_o \text{ are equivalent} \\ &\Leftrightarrow m \in \mathbf{S} \quad \text{by the construction of } \mathbf{S} \end{aligned}$$

We claim that \mathbf{S} is the linearization of \mathbf{H} , and \mathbf{H}' is the corresponding extension.

First, $E_{\mathbf{H}'-\mathbf{H}} = (\bigcup_o E_{(\mathbf{H}|_o)'}) \setminus E_{\mathbf{H}} = \bigcup_o (E_{(\mathbf{H}|_o)'} \setminus E_{\mathbf{H}|_o}) = \bigcup_o E_{(\mathbf{H}|_o)'-\mathbf{H}|_o}$. This is a set containing only response events.

Second, \mathbf{S} is sequential because:

1. The least event e^* of \mathbf{S} is also the least one of some \mathbf{S}_o . Since \mathbf{S}_o is a sequential history, e^* is an invocation event.
2. Let $m = \langle \text{inv}, \text{resp} \rangle$ be a complete method call. Suppose there exists a third event $e \in m'$ that $\text{inv} \prec_{\mathbf{S}} e \prec_{\mathbf{S}} \text{resp}$. Then m and m' are from different \mathbf{S}_o since each \mathbf{S}_o is a sequential history. However $\text{inv} \prec_{\mathbf{S}} e$ implies that $\text{inv} \prec_{\mathbf{H}} \text{inv}(m')$, then $\text{resp} \prec_{\mathbf{S}} e$ by the construction of \mathbf{S} . Such contradiction explains that the immediate successor of an invocation event must be its matching response event.
3. Let e be a response event and e' be its immediate successor in \mathbf{S} . Let m and m' be the two events' belonging method calls. If m and m' are from the same \mathbf{S}_o , then e' must be an invocation event since \mathbf{S}_o is sequential. If m and m' are from different \mathbf{S}_o and e' is a response event, $e \prec_{\mathbf{S}} e'$ implies that $\text{inv}(m) \prec_{\mathbf{H}} \text{inv}(m')$, then $e \prec_{\mathbf{S}} \text{inv}(m') \prec_{\mathbf{S}} e'$. Such contradiction means that the immediate successor of a response event must be an invocation event.

Third, \mathbf{S} is a legal history because \mathbf{S} is sequential and $\mathbf{S}|_o = \mathbf{S}_o$ which is the linearization of a single-object history.

Fourthly, for two different method calls m and m' , if $m \prec_{\text{complete}(\mathbf{H}')} m'$, then $m \prec_{\mathbf{S}} m'$:

1. Both m and m' are on the same object:

$$\begin{aligned}
 m \prec_{\text{complete}(\mathbf{H}')} m' &\Rightarrow \exists o ((m, m' \in \text{complete}(\mathbf{H}')|_o) \wedge (m \prec_{\text{complete}(\mathbf{H}')} m')) \\
 &\Rightarrow m \prec_{\text{complete}(\mathbf{H}')|_o} m' \\
 &\Rightarrow m \prec_{\text{complete}(\mathbf{H}'|_o)} m' \\
 &\Rightarrow m \prec_{\text{complete}((\mathbf{H}|_o)')} m' \\
 &\Rightarrow m \prec_{\mathbf{S}_o} m' \quad \text{since } \mathbf{S}_o \text{ is a linearization} \\
 &\Rightarrow m \prec_{\mathbf{S}} m'
 \end{aligned}$$

2. m and m' are on different objects:

$$\begin{aligned}
 m \prec_{\text{complete}(\mathbf{H}')} m' &\Rightarrow \exists o_i, o_j ((i \neq j) \wedge (m \in \text{complete}(\mathbf{H}')|_{o_i}) \\
 &\quad \wedge (m' \in \text{complete}(\mathbf{H}')|_{o_j}) \wedge (m \prec_{\text{complete}(\mathbf{H}')} m')) \\
 &\Rightarrow \text{inv}(m) \prec_{\text{complete}(\mathbf{H}')} \text{resp}(m) \prec_{\text{complete}(\mathbf{H}')} \text{inv}(m') \\
 &\Rightarrow \text{inv}(m) \prec_{\mathbf{H}} \text{inv}(m') \quad \text{by Proposition 2.13} \\
 &\Rightarrow \text{inv}(m) \prec_{\mathbf{S}} \text{resp}(m) \prec_{\mathbf{S}} \text{inv}(m') \prec_{\mathbf{S}} \text{resp}(m') \\
 &\Rightarrow m \prec_{\mathbf{S}} m'
 \end{aligned}$$

Finally, \mathbf{S} is equivalent to $\text{complete}(\mathbf{H}')$ because:

1. $\text{complete}(\mathbf{H}')|_p$ and $\mathbf{S}|_p$ are sequential and complete histories.
- 2.

$$\begin{aligned}
 m \langle i, j, x, y, -, - \rangle \in \text{complete}(\mathbf{H}')|_p &\Leftrightarrow (p_i = p) \wedge (m \langle i, j, x, y, -, - \rangle \in \text{complete}(\mathbf{H}')) \\
 &\Leftrightarrow (p_i = p) \wedge (m \langle i, j, x, y, -, - \rangle \in \mathbf{S}) \\
 &\Leftrightarrow m \langle i, j, x, y, -, - \rangle \in \mathbf{S}|_p
 \end{aligned}$$

- 3.

$$\begin{aligned}
 m \langle i, j, x, y, -, - \rangle \prec_{\text{complete}(\mathbf{H}')|_p} m' \langle r, s, u, v, -, - \rangle &\Leftrightarrow (m \langle i, j, x, y, -, - \rangle \prec_{\text{complete}(\mathbf{H}')} m' \langle r, s, u, v, -, - \rangle) \\
 &\quad \wedge (p_i = p_r = p) \\
 &\Leftrightarrow (m \langle i, j, x, y, -, - \rangle \prec_{\mathbf{S}} m' \langle r, s, u, v, -, - \rangle) \\
 &\quad \wedge (p_i = p_r = p) \\
 &\Leftrightarrow m \langle i, j, x, y, -, - \rangle \prec_{\mathbf{S}|_p} m' \langle r, s, u, v, -, - \rangle
 \end{aligned}$$

Now we complete the proof that \mathbf{S} is indeed the linearization of \mathbf{H} .

□

References

- [1] M. P. Herlihy and J. M. Wing, *Axioms for concurrent objects*, Technical Report CMU-CS-86-154, Carnegie Mellon Computer Sciences, 1986.
- [2] M. P. Herlihy and J. M. Wing, *Axioms for concurrent objects*, In Proceedings of the 14th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, pages 13-26, 1987.
- [3] M. P. Herlihy and J. M. Wing, *Linearizability: a correctness condition for concurrent objects*, ACM Transactions on Programming Languages and Systems, 12:463-492, 1990.
- [4] M. Herlihy and N. Shavit, *The Art of Multiprocessor Programming*, Morgan Kaufmann, 2008.
- [5] E. Szpilrajn. *Sur l'extension de l'ordre partiel*, Fundamenta Mathematicae, 16:386-389, 1930.
- [6] Herbert B. Enderton. *Elements of Set Theory*, Academic Press, New York, 1977.
- [7] Z. Manna and A. Pnueli. *Completing the temporal picture*, Theoretical Computer Science, 83(1):97-130, 1991.
- [8] C. Baiser and J-P Katoen. *Principles of Model Checking*, MIT Press, 2008.
- [9] L. Lamport. *The mutual exclusion problemPart I: A theory of interprocess communication*, Journal of the ACM (JACM), 33(2):313-326, 1986, ACM Press.
- [10] L. Lamport. *The mutual exclusion problemPart II: Statement and solutions*, Journal of the ACM (JACM), 33(2):327-348, 1986.
- [11] T. Kropf. *Introduction to Formal Hardware Verification*. Springer-Verlag, 1999.
- [12] Haoxiang Lin. *Well Extend Partial Well Orderings*. <http://arxiv.org/abs/1503.06514>, 2015.